**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №1**

**по дисциплине «Информатика»**

**Тема: Парадигмы программирования**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3341 |  | Кудин А.А. |
| Преподаватель |  | Иванов Д.В. |

Санкт-Петербург

2024

## Цель работы

Цель работы заключается в изучении и освоении концепций и техник объектно-ориентированного программирования (ООП), а также в получении практического опыта создания, использования и взаимодействия классов в программном коде.

## Задание

##### ​Базовый класс - транспорт Transport:

class Transport:

Поля объекта класс Transport:

* cредняя скорость (в км/ч, положительное целое число)
* максимальная скорость (в км/ч, положительное целое число)
* цена (в руб., положительное целое число)
* грузовой (значениями могут быть или True, или False)
* цвет (значение может быть одной из строк: w (white), g(gray), b(blue)).
* При создании экземпляра класса Transport необходимо убедиться, что переданные в конструктор параметры удовлетворяют требованиям, иначе выбросить исключение ValueError с текстом 'Invalid value'.

##### Автомобиль - Car:

class Car: #Наследуется от класса Transport

Поля объекта класс Car:

* cредняя скорость (в км/ч, положительное целое число)
* максимальная скорость (в км/ч, положительное целое число)
* цена (в руб., положительное целое число)
* грузовой (значениями могут быть или True, или False)
* цвет (значение может быть одной из строк: w (white), g(gray), b(blue)).
* мощность (в Вт, положительное целое число)
* количество колес (положительное целое число, не более 10)
* При создании экземпляра класса Car необходимо убедиться, что переданные в конструктор параметры удовлетворяют требованиям, иначе выбросить исключение ValueError с текстом 'Invalid value'.

*В данном классе необходимо реализовать следующие методы:*

Метод \_\_str\_\_():

Преобразование к строке вида: Car: средняя скорость <средняя скорость>, максимальная скорость <максимальная скорость>, цена <цена>, грузовой <грузовой>, цвет <цвет>, мощность <мощность>, количество колес <количество колес>.

Метод \_\_add\_\_():

Сложение средней скорости и максимальной скорости автомобиля. Возвращает число, полученное при сложении средней и максимальной скорости.

Метод \_\_eq\_\_():

Метод возвращает True, если два объекта класса равны, и False иначе. Два объекта типа Car равны, если равны количество колес, средняя скорость, максимальная скорость и мощность.

##### Самолет - Plane:

class Plane: #Наследуется от класса Transport

Поля объекта класс Plane:

* cредняя скорость (в км/ч, положительное целое число)
* максимальная скорость (в км/ч, положительное целое число)
* цена (в руб., положительное целое число)
* грузовой (значениями могут быть или True, или False)
* цвет (значение может быть одной из строк: w (white), g(gray), b(blue)).
* грузоподъемность (в кг, положительное целое число)
* размах крыльев (в м, положительное целое число)
* При создании экземпляра класса Plane необходимо убедиться, что переданные в конструктор параметры удовлетворяют требованиям, иначе выбросить исключение ValueError с текстом 'Invalid value'.

*В данном классе необходимо реализовать следующие методы:*

Метод \_\_str\_\_():

Преобразование к строке вида: Plane: средняя скорость <средняя скорость>, максимальная скорость <максимальная скорость>, цена <цена>, грузовой <грузовой>, цвет <цвет>, грузоподъемность <грузоподъемность>, размах крыльев <размах крыльев>.

Метод \_\_add\_\_():

Сложение средней скорости и максимальной скорости самолета. Возвращает число, полученное при сложении средней и максимальной скорости.

Метод \_\_eq\_\_():

Метод возвращает True, если два объекта класса равны по размерам, и False иначе. Два объекта типа Plane равны по размерам, если равны размах крыльев.

##### Корабль - Ship:

class Ship: #Наследуется от класса Transport

Поля объекта класс Ship:

* cредняя скорость (в км/ч, положительное целое число)
* максимальная скорость (в км/ч, положительное целое число)
* цена (в руб., положительное целое число)
* грузовой (значениями могут быть или True, или False)
* цвет (значение может быть одной из строк: w (white), g(gray), b(blue)).
* длина (в м, положительное целое число)
* высота борта (в м, положительное целое число)
* При создании экземпляра класса Ship необходимо убедиться, что переданные в конструктор параметры удовлетворяют требованиям, иначе выбросить исключение ValueError с текстом 'Invalid value'.

*В данном классе необходимо реализовать следующие методы:*

Метод \_\_str\_\_():

Преобразование к строке вида: Ship: средняя скорость <средняя скорость>, максимальная скорость <максимальная скорость>, цена <цена>, грузовой <грузовой>, цвет <цвет>, длина <длина>, высота борта <высота борта>.

Метод \_\_add\_\_():

Сложение средней скорости и максимальной скорости корабля. Возвращает число, полученное при сложении средней и максимальной скорости.

Метод \_\_eq\_\_():

Метод возвращает True, если два объекта класса равны по размерам, и False иначе. Два объекта типа Ship равны по размерам, если равны их длина и высота борта.

##### Необходимо определить список *list* для работы с транспортом: Автомобили:

class CarList – список автомобилей - наследуется от класса list.

Конструктор:

1. Вызвать конструктор базового класса.
2. Передать в конструктор строку name и присвоить её полю name созданного объекта

*Необходимо реализовать следующие методы:*

Метод append(p\_object): Переопределение метода append() списка. В случае, если p\_object - автомобиль, элемент добавляется в список, иначе выбрасывается исключение TypeError с текстом: Invalid type <тип\_объекта p\_object> (результат вызова функции type)

Метод print\_colors(): Вывести цвета всех автомобилей в виде строки (нумерация начинается с 1):

<i> автомобиль: <color[i]>

<j> автомобиль: <color[j]> ...

Метод print\_count(): Вывести количество автомобилей.

##### Самолеты:

class PlaneList – список самолетов - наследуется от класса list.

Конструктор:

1. Вызвать конструктор базового класса.
2. Передать в конструктор строку name и присвоить её полю name созданного объекта

*Необходимо реализовать следующие методы:*

Метод extend(iterable): Переопределение метода extend() списка. В случае, если элемент iterable - объект класса Plane, этот элемент добавляется в список, иначе не добавляется.

Метод print\_colors(): Вывести цвета всех самолетов в виде строки (нумерация начинается с 1):

<i> самолет: <color[i]>

<j> самолет: <color[j]> ...

Метод total\_speed(): Посчитать и вывести общую среднюю скорость всех самолетов.

##### Корабли:

class ShipList – список кораблей - наследуется от класса list.

Конструктор:

1. Вызвать конструктор базового класса.
2. Передать в конструктор строку name и присвоить её полю name созданного объекта

*Необходимо реализовать следующие методы:*

Метод append(p\_object): Переопределение метода append() списка. В случае, если p\_object - корабль, элемент добавляется в список, иначе выбрасывается исключение TypeError с текстом: Invalid type <тип\_объекта p\_object>

Метод print\_colors(): Вывести цвета всех кораблей в виде строки (нумерация начинается с 1):

<i> корабль: <color[i]>

<j> корабль: <color[j]> ...

Метод print\_ship(): Вывести те корабли, чья длина больше 150 метров, в виде строки:

Длина корабля №<i> больше 150 метров

Длина корабля №<j> больше 150 метров ...

В отчете укажите:

1. Изображение иерархии описанных вами классов.
2. Методы, которые вы переопределили (в том числе методы класса object).
3. В каких случаях будут использованы методы \_\_**str\_\_**() и \_\_**eq\_\_**().
4. Будут ли работать переопределенные методы класса list для CarList, PlaneList и ShipList? Объясните почему и приведите примеры.

## Выполнение работы

#### 1. Определение базового класса Transport

Создаем базовый класс Transport с полями для средней и максимальной скорости, цены, указания на грузовой транспорт, и цвета. При создании объекта класса проводится проверка валидности входных данных.

#### 2. Реализация класса Car

Класс Car наследуется от Transport и добавляет два новых поля: мощность и количество колес. Переопределяем методы \_\_str\_\_(), \_\_add\_\_() и \_\_eq\_\_() для выполнения специфичных операций для автомобилей.

#### 3. Реализация класса Plane

Аналогично, класс Plane наследуется от Transport и добавляет поля для грузоподъемности и размаха крыльев. Также переопределяем методы \_\_str\_\_(), \_\_add\_\_() и \_\_eq\_\_().

#### 4. Реализация класса Ship

Класс Ship, также наследующий Transport, вводит дополнительные поля для длины и высоты борта. Методы \_\_str\_\_(), \_\_add\_\_() и \_\_eq\_\_() переопределены специально для кораблей.

#### 5. Работа со списками транспортных средств

Для CarList, PlaneList, и ShipList реализуем методы добавления объектов в списки, проверяя их тип, а также специфические методы для работы с элементами этих списков, такие как печать цветов транспортных средств или подсчет общей средней скорости.

### Методы, которые были переопределены

* **Методы класса** object:
  + \_\_str\_\_(): Переопределен в классах Car, Plane, и Ship для предоставления читаемого текстового представления объектов этих классов.
  + \_\_eq\_\_(): Переопределен в классах Car, Plane, и Ship для сравнения объектов на основе определенных атрибутов.
* **Методы класса** list:
  + append(): Переопределен в классе CarList для добавления только объектов типа Car. В классе ShipList аналогично, но для объектов типа Ship.
  + extend(): Переопределен в классе PlaneList для добавления в список только объектов типа Plane.

### В каких случаях будут использованы методы \_\_str\_\_() и \_\_eq\_\_()

* \_\_str\_\_(): Этот метод будет использоваться каждый раз, когда необходимо получить строковое представление объекта, например, при печати объекта функцией print() или при преобразовании объекта в строку с помощью функции str(). Метод возвращает описание объекта в удобочитаемом виде.
* \_\_eq\_\_(): Этот метод используется для сравнения двух объектов на равенство. В контексте реализованных классов Car, Plane, и Ship, метод \_\_eq\_\_ позволяет сравнивать объекты на основе их специфических атрибутов, например, количество колес для Car, размах крыльев для Plane и комбинация длины и высоты борта для Ship.

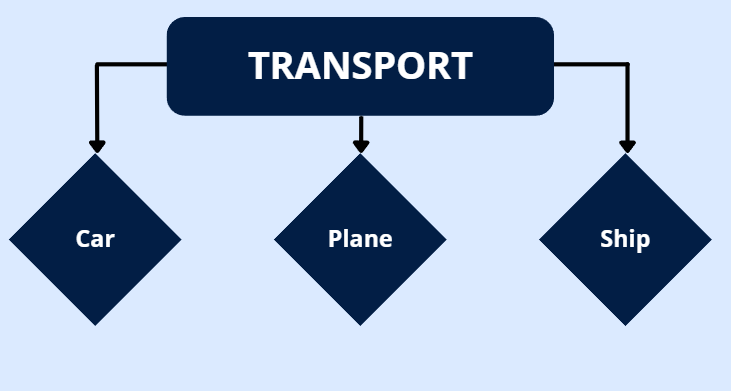
### Работа переопределенных методов класса list

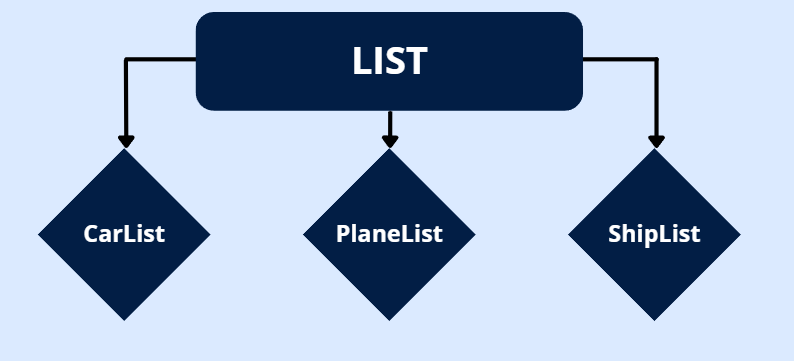
Переопределенные методы для классов CarList, PlaneList, и ShipList будут работать, поскольку они обеспечивают специализированное поведение для добавления элементов в список:

* В CarList и ShipList, метод append() теперь проверяет, соответствует ли объект добавляемого типа (автомобиль или корабль соответственно) перед добавлением в список. Это предотвращает добавление объектов неправильного типа.
* В PlaneList, метод extend() позволяет добавлять в список только объекты типа Plane, игнорируя объекты других типов.

Эти изменения гарантируют, что списки будут содержать только объекты соответствующих типов, что упрощает управление этими объектами и предотвращает ошибки типизации.

Иерархия классов





Разработанный программный код см. в приложении А.

## Тестирование

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | transport = Transport(70, 200, 50000, True, 'w') #транспорт  print(transport.average\_speed, transport.max\_speed, transport.price, transport.cargo, transport.color)  car1 = Car(70, 200, 50000, True, 'w', 100, 4) #авто  car2 = Car(70, 200, 50000, True, 'w', 100, 4)  print(car1.average\_speed, car1.max\_speed, car1.price, car1.cargo, car1.color, car1.power, car1.wheels)  print(car1.\_\_str\_\_())  print(car1.\_\_add\_\_())  print(car1.\_\_eq\_\_(car2))  plane1 = Plane(70, 200, 50000, True, 'w', 1000, 150) #самолет  plane2 = Plane(70, 200, 50000, True, 'w', 1000, 150)  print(plane1.average\_speed, plane1.max\_speed, plane1.price, plane1.cargo, plane1.color, plane1.load\_capacity, plane1.wingspan)  print(plane1.\_\_str\_\_())  print(plane1.\_\_add\_\_())  print(plane1.\_\_eq\_\_(plane2))  ship1 = Ship(70, 200, 50000, True, 'w', 200, 100) #корабль  ship2 = Ship(70, 200, 50000, True, 'w', 200, 100)  print(ship1.average\_speed, ship1.max\_speed, ship1.price, ship1.cargo, ship1.color, ship1.length, ship1.side\_height)  print(ship1.\_\_str\_\_())  print(ship1.\_\_add\_\_())  print(ship1.\_\_eq\_\_(ship2))  car\_list = CarList(Car) #список авто  car\_list.append(car1)  car\_list.append(car2)  car\_list.print\_colors()  car\_list.print\_count()  plane\_list = PlaneList(Plane) #список самолетов  plane\_list.extend([plane1, plane2])  plane\_list.print\_colors()  plane\_list.total\_speed()  ship\_list = ShipList(Ship) #список кораблей  ship\_list.append(ship1)  ship\_list.append(ship2)  ship\_list.print\_colors()  ship\_list.print\_ship() | 70 200 50000 True w  70 200 50000 True w 100 4  Car: средняя скорость 70, максимальная скорость 200, цена 50000, грузовой True, цвет w, мощность 100, количество колес 4.  270  True  70 200 50000 True w 1000 150  Plane: средняя скорость 70, максимальная скорость 200, цена 50000, грузовой True, цвет w, грузоподъемность 1000, размах крыльев 150.  270  True  70 200 50000 True w 200 100  Ship: средняя скорость 70, максимальная скорость 200, цена 50000, грузовой True, цвет w, длина 200, высота борта 100.  270  True  1 автомобиль: w  2 автомобиль: w  2  1 самолет: w  2 самолет: w  140  1 корабль: w  2 корабль: w  Длина корабля №1 больше 150 метров | Тестирование созданных классов, их методов (созданных и переопределенных). |

## Выводы

В ходе выполнения работы были рассмотрены и реализованы ключевые аспекты объектно-ориентированного программирования (ООП), что позволило углубить понимание принципов создания и использования классов и объектов.

# Приложение А Исходный код программы

Название файла: main.py

class Transport:

def \_\_init\_\_(self, average\_speed, max\_speed, price, cargo, color):

if not all([isinstance(average\_speed, int), average\_speed > 0,

isinstance(max\_speed, int), max\_speed > 0,

isinstance(price, int), price > 0,

isinstance(cargo, bool),

color in ['w','g','b']]):

raise ValueError('Invalid value')

self.average\_speed = average\_speed

self.max\_speed = max\_speed

self.price = price

self.cargo = cargo

self.color = color

class Car(Transport):

def \_\_init\_\_(self, average\_speed, max\_speed, price, cargo, color, power, wheels):

super().\_\_init\_\_(average\_speed, max\_speed, price, cargo, color)

if not (isinstance(power, int) and power > 0 and isinstance(wheels, int) and 0 < wheels <= 10):

raise ValueError('Invalid value')

self.power = power

self.wheels = wheels

def \_\_str\_\_(self):

return (f"Car: средняя скорость {self.average\_speed}, максимальная скорость {self.max\_speed}, "

f"цена {self.price}, грузовой {self.cargo}, цвет {self.color}, мощность {self.power}, "

f"количество колес {self.wheels}.")

def \_\_add\_\_(self):

return self.average\_speed + self.max\_speed

def \_\_eq\_\_(self, other):

return (self.wheels == other.wheels and

self.average\_speed == other.average\_speed and

self.max\_speed == other.max\_speed and

self.power == other.power)

class Plane(Transport):

def \_\_init\_\_(self, average\_speed, max\_speed, price, cargo, color, load\_capacity, wingspan):

super().\_\_init\_\_(average\_speed, max\_speed, price, cargo, color)

if not (isinstance(load\_capacity, int) and load\_capacity > 0 and isinstance(wingspan, int) and wingspan > 0):

raise ValueError('Invalid value')

self.load\_capacity = load\_capacity

self.wingspan = wingspan

def \_\_str\_\_(self):

return (f"Plane: средняя скорость {self.average\_speed}, максимальная скорость {self.max\_speed}, "

f"цена {self.price}, грузовой {self.cargo}, цвет {self.color}, грузоподъемность {self.load\_capacity}, "

f"размах крыльев {self.wingspan}.")

def \_\_add\_\_(self):

return self.average\_speed + self.max\_speed

def \_\_eq\_\_(self, other):

return self.wingspan == other.wingspan

class Ship(Transport):

def \_\_init\_\_(self, average\_speed, max\_speed, price, cargo, color, length, side\_height):

super().\_\_init\_\_(average\_speed, max\_speed, price, cargo, color)

if not (isinstance(length, int) and length > 0 and isinstance(side\_height, int) and side\_height > 0):

raise ValueError('Invalid value')

self.length = length

self.side\_height = side\_height

def \_\_str\_\_(self):

return (f"Ship: средняя скорость {self.average\_speed}, максимальная скорость {self.max\_speed}, "

f"цена {self.price}, грузовой {self.cargo}, цвет {self.color}, длина {self.length}, "

f"высота борта {self.side\_height}.")

def \_\_add\_\_(self):

return self.average\_speed + self.max\_speed

def \_\_eq\_\_(self, other):

return self.length == other.length and self.side\_height == other.side\_height

class CarList(list):

def \_\_init\_\_(self, name):

super().\_\_init\_\_()

self.name = name

def append(self, p\_object):

if not isinstance(p\_object, Car):

raise TypeError(f'Invalid type {type(p\_object).\_\_name\_\_}')

super().append(p\_object)

def print\_colors(self):

for i, car in enumerate(self, start=1):

print(f"{i} автомобиль: {car.color}")

def print\_count(self):

print(f"{len(self)}")

class PlaneList(list):

def \_\_init\_\_(self, name):

super().\_\_init\_\_()

self.name = name

def extend(self, iterable):

for item in iterable:

if not isinstance(item, Plane):

continue

super().append(item)

def print\_colors(self):

for i, plane in enumerate(self, start=1):

print(f"{i} самолет: {plane.color}")

def total\_speed(self):

total = sum(plane.average\_speed for plane in self)

print(f"{total}")

class ShipList(list):

def \_\_init\_\_(self, name):

super().\_\_init\_\_()

self.name = name

def append(self, p\_object):

if not isinstance(p\_object, Ship):

raise TypeError(f'Invalid type {type(p\_object).\_\_name\_\_}')

super().append(p\_object)

def print\_colors(self):

for i, ship in enumerate(self, start=1):

print(f"{i} корабль: {ship.color}")

def print\_ship(self):

for i, ship in enumerate(self, start=1):

if ship.length > 150:

print(f"Длина корабля №{i} больше 150 метров")